BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

APPLICATION FOR VISUALIZING ANIMATED 3D OBJECTS USING AUGMENTED REALITY ON IOS

APLIKACE PRO VIZUALIZACI ANIMOVANÝCH 3D OBJEKTŮ S VYUŽITÍM ROZŠÍŘENÉ REALITY NA IOS

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR **MARTIN MINÁRIK**
AUTOR PRÁCE

SUPERVISOR **Ing. VÍTĚZSLAV BERAN, Ph.D.**
VEDOUCÍ PRÁCE

BRNO 2019

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2018/2019

Zadání bakalářské práce

Student: **Minárik Martin**
Program: Informační technologie
Název: **Aplikace pro vizualizaci animovaných 3D objektů s využitím rozšířené reality na iOS**
Application for Visualizing Animated 3D Objects Using Augmented Reality on iOS
Kategorie: Uživatelská rozhraní

Zadání:

1. Prostudujte možnosti technologie ARKit iOS. Seznamte se s vhodnými nástroji vizualizace animovaných 3D objektů na iOS.
2. Navrhněte systém pro vizualizace animovaných 3D objektů v prostředí iOS, včetně interaktivnáho rozhraní umožňující umístění objektu do scény. Zvažte možnosti využití vodicích vizuálních značek.
3. Implementujte navržený systém s využitím relevantních dostupných technologií. Zaměřte se na efektivitu uživatelské interakce.
4. Vyhodnoťte vlastnosti výsledného systému na základě experimentů v reálném prostředí.
5. Prezentujte klíčové vlastnosti řešení formou plakátu a krátkého videa.

Literatura:

- Dieter Schmalstieg, Tobias Hollerer. *Augmented Reality: Principles and Practice.* Addison-Wesley, 2016. ISBN: 978-0321883575.
- Russ Unger, Carolyn Chandler. A Project Guide to UX Design: For user experience designers in the field or in the making. New Riders, 2012. ISBN: 0132931729.
- Dále dle pokynu vedoucího.

Pro udělení zápočtu za první semestr je požadováno:

- Body 1, 2 a částečně bod 3.

Podrobné závazné pokyny pro vypracování práce viz http://www.fit.vutbr.cz/info/szz/

Vedoucí práce: **Beran Vítězslav, Ing., Ph.D.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2018
Datum odevzdání: 15. května 2019
Datum schválení: 6. listopadu 2018

Abstract

The aim of this thesis is to create an application for visualizing 3-D animated objects in augmented reality on a device with operating system iOS. The work demonstrates how to load an object to display within the augmented reality as well methods for animating, changing the state and manipulating the objects. The thesis describes preparation of the object for visualization, inner structure of the object and animation files as well as supported files. The result of the thesis is an application using ARKit framwerok allowing user to display and manipulate the objects capable of animating in augmented reality.

Abstrakt

Cieľom tejto práce je vytvorenie aplikácie pre vizualizáciu animovaných 3-D objektov za pomoci rozšírenej reality na zariadeniach s operačným systémom iOS. Práca demonštruje, ako načítať objekt pre zobrazenie v rozšírenej realite, ako aj metódy pre jeho animovanie, zmenu stavu a manipuláciu s objektom. Práca sa zaoberá prípravou objektu, pre zobrazenie, vnútornou štruktúrou súborov s objektami a animáciami a popisom podporovaných formátov. Výsledkom práce je samotná aplikácia naprogramovaná pomocou frameworku ARKit, ktorá umožňuje zobraziť a manipulovať s objektami s možnosťou animácie v rozšírenej realite.

Keywords

augmented reality, iOS, ARKit, SceneKit

Klíčová slova

rozšírená realita, iOS, ARKit, SceneKit

Reference

MINÁRIK, Martin. *Application for Visualizing Animated 3D Objects Using Augmented Reality on iOS*. Brno, 2019. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Vítězslav Beran, Ph.D.

Rozšířený abstrakt

Cieľom bakalárskej práce je vytvorenie aplikácie umožňujúcej vizualizáciu 3-D animovaných objektov za pomoci rozšírenej reality na zariadeniach s operačným systémom iOS a možnosti manipulácie s objektmi. Práca popisuje a rieši problém zobrazenia 3-D objektov, ktoré nemajú formát kompatibilný so zobrazením pomocou natívnych frameworkov ARKit a SceneKit. Teoretická časť práce popisuje využitie rozšírenej reality, aktuálne existujúce riešenia, popis algoritmov, ktoré sa používajú pri vizualizácii a mapovaní rozšírenej reality, ako aj detailný popis frameworkov a porovnanie ARKitu s ARCore.

Vlastné riešenie problému je vytvorenie aplikácie, ktorá dokáže zobrazovať 3-D animované objekty. Aplikácia využíva hlavne frameworky ARKit a SceneKit. ARKit sa využíva pre zobrazenie a ukotvenie objektu v reálnom prostredí zachytenom kamerou mobilného zariadenia. Využíva pri tom vizuálnu inerciálnu odometriu, čo znamená informácie z kamery a senzorov zariadenia pre spracovanie okolitého sveta. SceneKit aplikácia využíva ako vykreslovací engine pre zobrazenie, ale taktiež animáciu samotných 3-D objektov.

Práca rieši prevod objektov, ktoré môžu už byť vytvorené a nemajú špecifický formát súboru podporovaný v SceneKite. Podporované explicitné, okrem implicitného Scene sú dva formáty — Alembic alebo COLLADA. Pretože SceneKit podporuje iba špecifickú vnútornú štruktúru formátu, samotné konvertovanie súboru nestačí. Preto bol vytvorený spôsob ako odstrániť nepotrebné elementy a koretne zobraziť súbor pre potreby programovacieho IDE aplikácií pre iOS — Xcode. Kvôli problémom pri načítaní animačných súborov zo súborov obsahujúcich aj model aj animáciu sa tieto delia na dva. Jeden súbor obsahuje 3-D model vrátane štruktúry kostí a druhý animáčný súbor taktiež so štruktúrou kostí. Tieto súbory sa následne môžu použiť na načítanie v aplikácii. Načítanie objektov je uskutočnené pomocou statickej funkcie, ktorá prehľadáva špecifickú zložku v zdrojovej zložke aplikácie. Načítanie animácií prebieha pomocou rozšírenia API pre prehrávania animácií, ktoré skenuje štruktúru súboru pre možné animácie a vracia zhodu.

Grafické rozhranie aplikácie je zložené z jednoduchého GUI, ktoré tvoria 2 tlačidlá — na pridávanie objektov, ktoré zobrazí tabuľku so všetkými objektami vhodnými na pridanie a tlačidlom, ktoré uvedie aplikáciu do odstraňovacieho módu.

Pre interakciu s objektom boli implementované metódy, ktoré umožňujú premiestňovať, zväčšovať, otáčať, animovať a odstrániť objekt. Tieto fungujú na základe rozoznávania gest a následným hit testingom zasiahnutých objektov v scéne.

Okrem tejto aplikácie bola spolu s Adamom Jurczykom v rámci bakalárskej práce implementovaná demo aplikácia Rytier, ktorá umožňuje zobrazovať rovnaký animovaný 3-D objekt na viacerých zariadeniach zároveň pomocou multipeer connectivity v rámci ARKitu 2. Z ďalších funkcionalít aplikácie sú tam implementované možnosť interakcie s rytierom, ktorý sa zobrazuje v reálnom čase na všetkých zariadeniach pripojených v multipeer session. Táto aplikácia obsahuje špecifický model a možnosťami animácie a menenia konfigurácie objektu pri znalosti štruktúry modelu.

V bakalárskej práci bola naimplementovaná aplikácia pre vizualizáciu animovaných 3-D objektov. Aj napriek tomu, že táto aplikácia sama o sebe nie je finálny produkt, ktorý by sa dal pridať do obchodu na stiahnutie, má na to potenciál do budúcna. Pri pridaní možnosti spojenia s cloudom a zdieľania scény s inými zariadeniami má potenciál stať sa unikátnou aplikáciou vhodnou pre marketing.

Application for Visualizing Animated 3D Objects Using Augmented Reality on iOS

Declaration

I declare that I have prepared this Bachelors thesis independently, under the supervision of Ing. Vítězslav Beran, Ph. D. I listed all of the literary sources and publications that I have used.

.....
Martin Minárik
May 16, 2019

Acknowledgements

I would like to thank my supervisor Ing. Vítězslav Beran, Ph.D. for his help and patience during the creation of this thesis. I would also like to thanks Adam Jurczyk for cooperation and patience while working on the project.

Contents

Chapter 1

Introduction

The main goal of this bachelor thesis is to create an application that provides visualization and interaction with 3-D virtual objects in augmented reality on devices running iOS. Besides this functionality, thesis should also describe the preparation of the 3-D object in unsupported format for use in the application.

First part of the thesis is dedicated to the augment reality. Due to mobile devices being more technologically advanced, the uses of augmented reality diverted from expensive dedicated devices into wide-spread mobile devices. At the start examples of augmented reality applications for smart devices and use-cases of augmented reality in general are described, following methods for visualization and mapping in augmented reality.

To really understand how to easily create an application using augmented reality on iOS, the knowledge of ARKit is essential. Parts of the framework and some key classes and methods are explained in the theoretical part of the thesis. Displaying of the virtual objects in augmented reality works the best with SceneKit, graphic rendering engine for iOS. In the thesis is an explanation on how does that SceneKit work, especially when used together with ARKit. SceneKit section also talks about supported implicit and explicit formats that are part of understanding the compatibility issues.

The key design elements together with ways of interaction with an object are defined in the design section. Consecutively, the section contains information about the conversion process and preparation of the animation and object files with explanation as why were they chosen.

An application demonstrating multipeer connectivity and displaying animatable object on multiple devices, a part of the Excel@FIT conference, is shown and it's potential explained in the end of design section.

Because the final application is universal, implementation and testing of the application was done on both tablet Apple iPad 6 and Apple iPhone 6s. In the thesis, there are listed interesting code snippets together with some of the key implementation parts of the project.

Chapter 2

Augmented Reality

Augmented reality has many definitions but they all share the same characteristics. One of the most recognized definitions is one written by Ronald T. Azuma (1997). Augmented reality should follow these 3 characteristics:

1. Combines real and virtual
2. Interactive in real time
3. Registered in 3-D

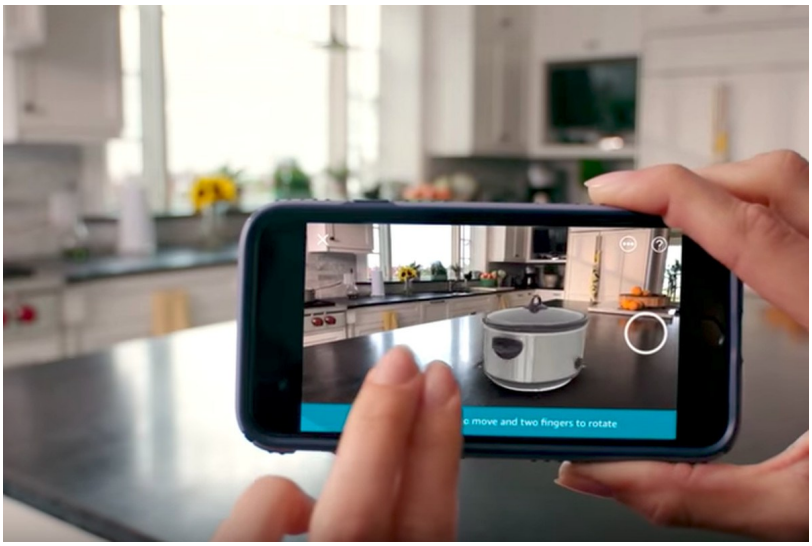

Figure 2.1: An example how Amazon's AR View let's online shoppers see how a product will look in their homes. *Source:* [16]

As the definition states, augmented reality supplements real world with virtual objects but still keeps the real world environment. Even though the definition doesn't specify the technology, this doesn't mean that film overlays added post production are augmented reality but many people still think so. Such films feature photo realistic virtual objects, however they lack interactivity with virtual objects in real life in 3-D space. As seen on Figure 2.1, the pot is inserted in real world environment and combined with kitchen table. It looks as if it was really there, object doesn't float in the air but rather sits on the table.

To get the best user experience, technologies should react to user's actions within the fastest time period possible.

Since augmented reality is a variation of virtual environments, or commonly virtual reality, many people mistake the two. Virtual reality surrounds the user with synthetic environment. These technologies usually try to stimulate more than just one sense by adding sound effects, vibration and more. This means that users are separated from and can't see the real world surrounding them. On the other hand, virtual reality only adds a layer on top the reality.

2.1 History

First annotations of computer-generated layer over the real world were introduced in 1960s by American scientist Ivan Sutherland. First, he wrote an essay The Ultimate Display discussing the matter, later in 1968 he constructed his first head-mounted three-dimensional display [21]. It was based on computer with graphic hardware that could render 3-D objects over real world using goggles. Even though it was mounted to the ceiling because of its weight, it was still a good starting point to amplify the research.

The term augmented reality first appeared in the work of Caudell and Mizell at company Boeing in 1992. This was marked as the birth of the term „augmented reality". This was also the first time augmented reality was used for industrial purposes. It was used as a guide for the wire assembly for aircrafts due to low efficiency of the workers.

A lot of progress was made during the 90s. For example, first handheld spatially aware display, an application allowing to view inside the womb of an expecting mother, which is still challenging to create today, first multi-peer user sharing augmented reality system. There was an effort to create wearable and handheld technologies that could be used even outside.

In 1999, ARToolKit, the first open-source software platform for AR was developed by Kato and Billinghurst. Using ARToolKit library, users were able to track symbols that could be easily printed out.

During the 21st century, as the research in mobile computing evolved, AR expanded to this platform. In 2008, first real tracking systems for smartphones were introduced. While smartphones aren't the best choice for AR, many developers use it for its availability and low cost of hardware. Basic users don't have to buy additional devices, they can use the smartphone that they already own. Aside from smartphones, there were introduced a lot of augmented reality glasses and headsets.

2.2 Fields of use

The use of augmented reality has expanded into many fields during past years. It isn't only limited to professional use anymore. Thanks to widespread mobile devices with advanced technology it has found use even in day-to-day lives of regular people. As the number of users grew, there were also new use cases. It is only excepted that there will be new possibilities using augmented reality in the future.

Manufacturing and repair

AR is heavily used when it comes to complicated processes with machines that have extensive manuals which are hard to understand. Especially when the steps must be followed

exactly. AR makes this easier with instructions that are displayed in real-world as an overlay showing step-by-step the tasks that need to be done. These guides in form of virtual objects are used to reduce mistakes, speed up the tasks and improve the clarity of instructions via AR.

In space NASA launched project Sidekick that helps astronauts complete maintenance and other work tasks that are critical [6]. Since time in space is extremely valuable, tasks and maintenance instructions must be unambiguous. The first part of the system displays detailed visual instructions on the astronauts' glasses, guiding them to perform the necessary procedures in the right order. The second part let's remote technologists guide the astronauts through repairs and experiments in real-time using a Skype up-link. These experts can also annotate on the environment to explain the problem without the need to navigate astronauts using voice. Technologist are seeing the same environment as they peer through the HoloLens.

Figure 2.2: Instructional overlay for maintenance. *Source:* [20]

Medicine

In this field takes advantage of imaging technology. Some areas of medicine are transforming thanks to visualizing ability of showing different parts of body in real-time. Thanks to technologies like Google Glass, surgeons can have 3-D anatomical information overlaid on top of the patient during the surgery. This significantly helps with showing vital anatomical structures which are scanned prior to the operation and then highlighted in AR. Major blood vessels can be color-coded.

Figure 2.3: AR application displaying blood vessels on an iPad during liver surgery. *Source:* [12]

Such technologies use electromagnetic field and tracking of surgeon's movements together with AR. Other than that, surgeon can live-stream and get real-time feedback from more experienced doctors anywhere in the world. This can prevent some medical errors made by even experienced surgeons.

AR can be used with X-rays and body scans to detect previous health state of the patient, to find previous injuries without having to at the desktop [5]. The biggest advantage is that surgeon can look directly at the operation without having to concentrate on the monitor.

Personal Information Display

This fields is aimed towards mass audiences and tries to help people with everyday tasks using AR. Apps like Yelp Monocle rely on smartphones' GPS, compass readings, or image recognition to display user places of interest. Interesting use of AR is application Google Translate that superimposes translations of text over the actual text, recognized in real time.

Applications like Snapchat or Instagram use AR to detect user's face and add overlay filter. These uses on smartphones are becoming more common and users usually don't even know that they are dealing with AR.

Figure 2.4: Visualizing places of interest in AR browser. *Source:* [10]

Navigation

Navigation using AR was used for the first time in military aircrafts and helicopters. They provided navigation, flight information, and they could register targets in the environment which helped with aiming weapons. Nowadays, as the geo-information had been improved, navigation using AR can even overlay roadblocks. AR navigation can be used on a smartphone using applications like Sygic. However, there should be heads-up displays (HUDs) displays with this feature built into cars in near future. There are many concepts, but the most promising so far is WayRay unveiled at the 2019 Consumer Electronics Show [9].

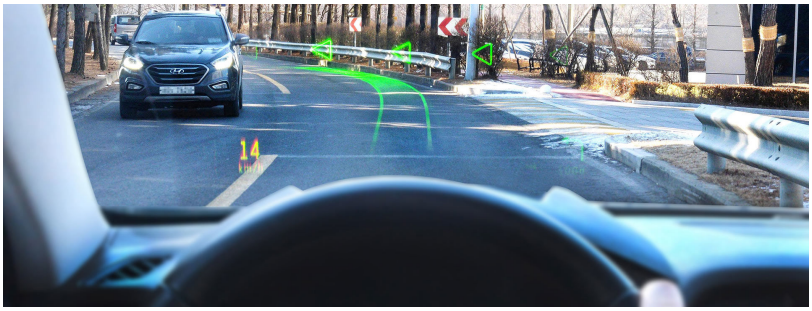

Figure 2.5: An demonstration of WayRay equipped in the Genesis G80. *Source:* [9]

Advertising and Commerce

Marketing through AR offer whole new level of user experience. Displaying products in user's home or in a showroom and personalizing it to given preferences changes the user's perception of the product way easier than words [15]. The most popular app that let's users display furniture anywhere they want is Ikea Place[1] application. Other common use is displaying virtual content over recognized images in newspaper, or some other media. When user scans specific still image, video starts playing, or parts of the image grow into 3-D model with annotations.

Figure 2.6: Visualizing furniture using Ikea application. *Source:* [11]

Gaming

AR games are popular and commonly played on smartphones, tablets and portable gaming systems. One of the first games supporting AR was The Eye of Judgment for the Sony PlayStation 3 [18]. AR gaming uses the existing environment and creates a playing field within it. There could be characters climbing from sofas to coffee tables. These games take a lot of time to develop. On the other hand, there are games that are still using AR but aren't interacting with environment as much. Example of such game should chess that only detects plane and places chessboard on top. In general, gaming using AR usually

[1] https://itunes.apple.com/us/app/ikea-place/id1279244498?mt=8

superimposes a precreated environment on top a user's environment taking advantage of the diversity of the real-world to keep the games interesting.

Figure 2.7: Slingshot made by Apple Inc. An AR game that allows multiple users to play at against each other. *Source:* [19]

Military

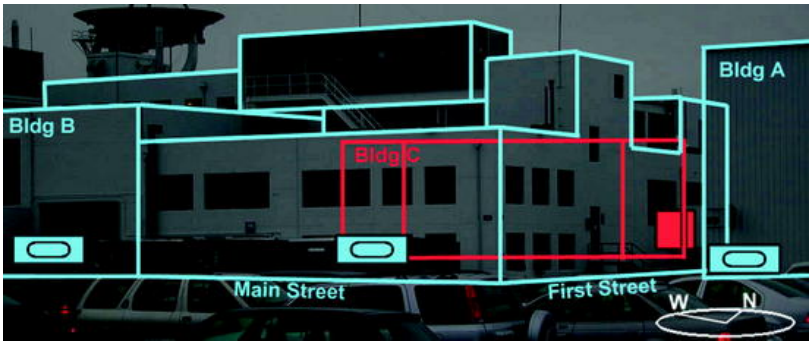

Figure 2.8: Displaying strategic points and buildings for military use. *Source:* [14]

Military field has long history of AR used in many scenarios. Most notable uses are in situation awareness to keep track past, present, and future of forces operating in an environment. Other task of AR is to add appropriate information for user in the team's mission. This means that instead of knowing every information which would be hard to process use would be aware only of various tasks, plans and roles that may be fulfilled by the particular user at a given time. This information must be present at exactly right time so that user can detect for example whether an approaching vehicle is a threat, or squad can come to the aid of another unit. Because of these use-cases, military problems are much more difficult and complex than the civilian applications. [14].

Education

AR is still new to the education but so far the use of AR has received positive feedback from students. It is easier to engage students in learning process and improve their understanding of what are they taught. There are many subjects which can be used with the addition of AR. For example, in chemistry to provide better understanding of molecules and spatial relation between them, in mathematics to teach geometry using 3-D virtual geometrical concepts, in history to enhance the experience of the visitors of museums [17].

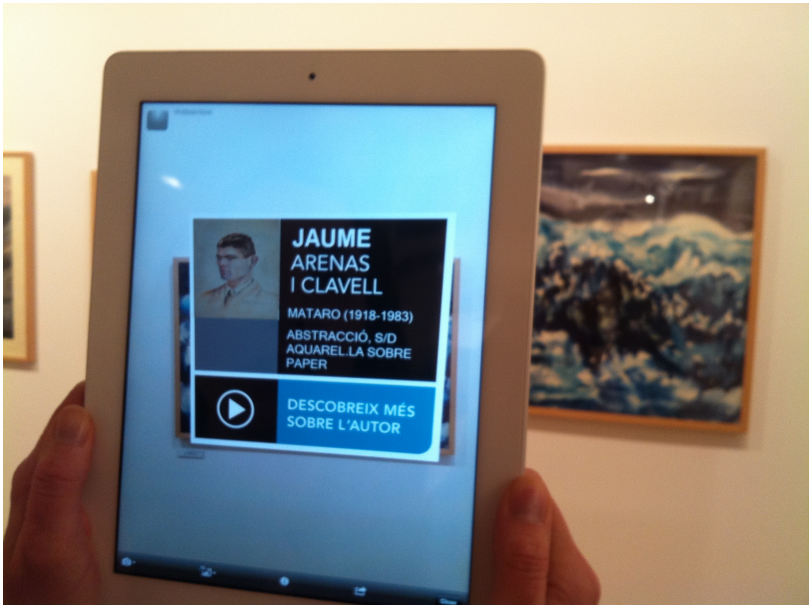

Figure 2.9: Additional information shown on an iPad after detecting the painting. *Source:* [1]

2.3 Existing solutions

There are quite a few applications providing visualization of 3-D objects in the AR. The most notable one is the Ikea Place app. Application scans the environment and let's the user choose where to place an object from the Ikea shop. Objects are limited to only models from their website and the users can add them to their cart and order them. Ikea isn't the only application that provides such experience, but it is the most popular. There are applications like V-3Display[2] or AR Display[3], nevertheless they don't perform as well.

Similarly, Civilisations AR[4] displays 3-D artefacts. This application provides educational features like narration. Providing full 3-D model that can be rotated or scaled brings new experience almost as if the user was able to hold the artefacts in hands.

Vuforia Chalk[5] offers an AR remote assistance. The application sets up a video call between two devices, where the person getting advice shows their vision using rear camera

[2] https://itunes.apple.com/us/app/v-3display/id1435607299?mt=8
[3] https://itunes.apple.com/us/app/ar-display/id1257661182?mt=8
[4] https://itunes.apple.com/us/app/civilisations-ar/id1350792208?mt=8
[5] https://itunes.apple.com/us/app/vuforia-chalk/id1280738776?mt=4

to the person giving advice, who then draws their suggestions that are mapped and stick onto objects in the actual environment.

2.4 Visualizing

Six degrees of freedom (6DOF)

Degree of freedom is an independent dimension of measurement defining the ways that object can move in space. There are total of six degrees of freedom determining the pose of an object in three-dimensional space. 6DOF are divided into two categories each having three degrees of freedom. Firstly, three degrees of freedom for position tracking X, Y and Z-axis translation. Secondly, three degrees of freedom for orientation tracking pitch, yaw, or roll (rotation). Some sensors only work with one category, so 3DOF. Those can't provide the full AR experience at all times. For example, they can display AR content if the device is not moving but the content moves out of its position as soon as the device moves out of its position.

Visual Inertial Odometry

VIO means that device tracks its 6DOF position in real-time In ARKit, 3DOF used for position are measured by visual system that matches user's position with a point in real world using camera sensor which is updated each frame. 3DOF used for orientation are determined by IMU or inertial measurement unit. IMU in ARKit are device's accelerometer and gyroscope. This information forming user's pose is calculated twice at the same time in parallel.

Simultaneous localization and mapping

Simultaneous localization and mapping or SLAM (sometimes called Visual Odometry) tries to solve a problem of a truly autonomous robot. This means that if the robot was sent into an unknown environment without any previous knowledge, he would build a map of the environment as well as simultaneously determining its own position within the map. However, this map is only used to support the incremental tracking [7].

A basic SLAM pipeline encompasses the following steps [18]:

1. Detect interest points in the first frame—for example, using Harris or FAST corners (see section 2.4).
2. Track the interest point in 2D from the previous frame—for example, using KLT (see section 2.4).
3. Determine the essential matrix between the current and previous frames from the feature correspondences with a five-point algorithm (see section 2.4) inside a RANSAC loop.
4. Recover the incremental camera pose from the essential matrix.
5. Since the essential matrix determines the translation part of the pose only up to scale, this scale must be estimated separately, so that it is consistent throughout the tracked image sequence. To achieve this aim, 3D point locations are triangulated

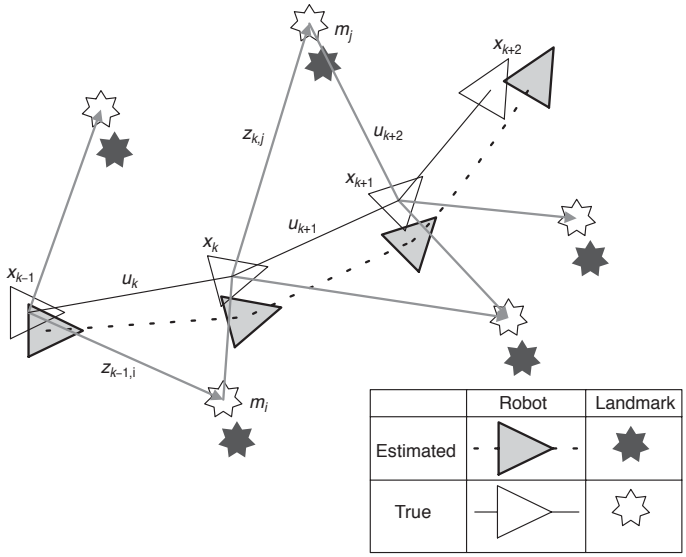

Figure 2.10: A simultaneous estimate of both robot and landmark locations is required. The true locations are never known or measured directly. Observations are made between the robot and landmark locations. *Source:* [7]

from multiple 3D observations of the same image feature over time. This approach is called structure from motion (SFM).

6. Proceed to the next frame.

Harris Corners

There must be a strong gradient in both horizontal and vertical dimensions to detect a point in 2-D. Therefore, suitable interest points will be shaped like circular blobs or corners. Harris detector uses auto-correlation to determine corners.

Features From Accelerated Segment

Features From Accelerated Segment or FAST is detector that is optimized for speed. Because of it's high-speed performance, FAST is suitable for real-time video processing applications even on mobile devices. Detector uses a discretized circle centered on the candidate point. If there exists a contiguous arc of pixel with sufficient contrast to the center pixel, which covers up to three quarters of a circle, a point is classified as a corner.

Rosten and Drummond improved FAST with machine learning that creates a decision tree for determining the order of the arc pixels to be tested. The goal of this improvement was to finish the testing as soon as possible.

There are different types of FAST algorithm. Their names are derived from arc length in pixels. The most used is the version with machine learning, especially for arc lengths smaller than 12. If the machine learning isn't used, FAST12 is the most common.

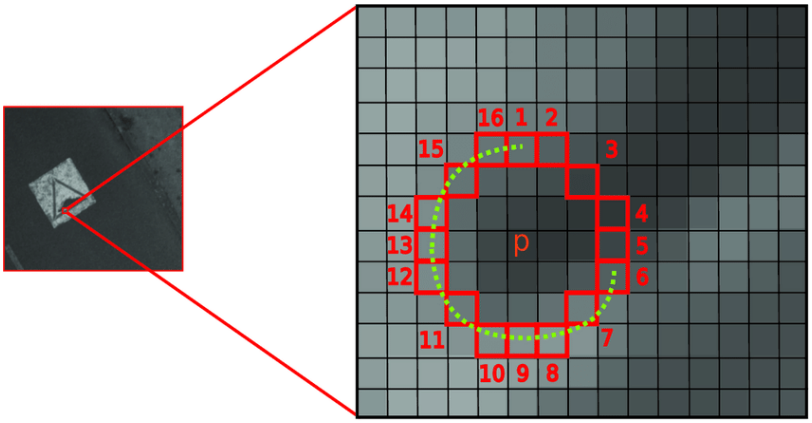

Figure 2.11: Features from Accelerated Segment Test (FAST)-12. *Source:* [4]

Weaknesses of FAST detector are limited robustness to noise and motion blur. These can fairly easily make FAST unusable in special scenarios.

Kanade-Lucas-Tomasi Tracking

Kanade-Lucas-Tomasi tracker extracts points from an initial image and then tracks them using optical flow. Tracking finds parameters of a warp that transforms a template image into the input image. The warp will usually be restricted to an affine transformation, which is sufficient enough to model the deformation of an image patch observed after small camera motion. For such small, incremental motions, affine transformations are very similar to perspective disortion effects, which wold be much more expensive to compute.

Five-Point Algorithm for Essential Matrix

Algorithm determines the relative camera motion from 2-D point correspondences, Nistér's algorithm computed from five point correspondences. Relative pose between the camers can be calculated using SVD. If the optical centers of two camera viewpoints aren't different, the implicit triangulation is ill-defined. Because of this, SLAM cameras have distinct travelling motion and algorithm fails only when the camera is rotated.

Chapter 3

Augmented reality frameworks for smartphones

This chapter will discuss the frameworks for AR on smartphones that are used in this project as well as some notable ones that aren't.

3.1 ARKit

ARKit[1] is framework made by Apple Inc. which works on operating system iOS and makes development of AR applications for iPhone and iPad easier. ARKit was released in 2017 with iOS 11 version and supports devices that are using A9 or newer processors.

Since Apple doesn't share its source codes, there isn't much information about methodologies they use. However, Apple revealed that ARKit is using Visual Inertial Odometry or VIO (see section 2.4). VIO combines data from AVFoundation and CoreMotion to get device's 6DOF. This means that the framework doesn't need any external sensors, or trackers, or any prior knowledge about the environment.

ARKit's Functions

ARKit generates information that can be used with many rendering engines. The easiest to integrate are SceneKit for 3-D graphics and SpriteKit for 2-D graphics. Supported are also Metal, Unity, and Unreal.

ARKit has integrated plane detection which allows to overlay objects onto the real-world as if they were lying in real-world. Since version 1.5, ARKit is able to scan both horizontal and vertical planes. Furthermore, the planes don't have to be just 4-point polygons but more complex shapes. This leads to better recognition of more complex planes. To be more realistic, ARKit has function that estimates the ambient light intensity as well as the ambient color temperature. If the device has a camera that can monitor depth of the scene, ARKit applications have better detection and can detect position in scene, topology and facial expressions.

Detection of reference images adds recognition of 2-D images in the scene [2]. The pictures used for detection should have detectable features. When recognized, application can trigger many interesting functions as seen in the section 2.2.

[1] https://developer.apple.com/documentation/arkit/

In 2018, Apple Inc. has released ARKit 2 that allows users share the same virtual experience on multiple devices at the same time. Virtual objects could be persistent or saved and loaded at the later time. With iOS 12 there was a new ability introduced that allows to measure real-world objects using ARKit.

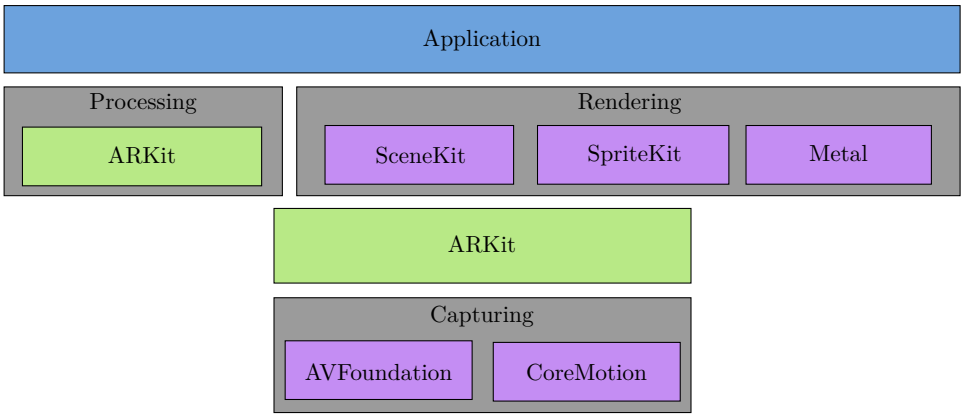

Figure 3.1: Structure of ARKit framework in context of application.

ARSession

`ARSession`[2] is the main ARKit class that manages the device camera and motion processing needed for AR. Device is getting data `AVFoundation` and `CoreMotion` that are collected from camera and motion sensing hardware. `ARSession` analyses the data and uses the results to establish a connection between the real-world and AR content. Depending on which rendering engine is used, `ARSession` object can be built different ways. If SpriteKit or SceneKit is used, `ARSession` object is part of `ARSKView` or ARSCNView. If Metal is used, the developer must create his own renderer and maintain the `ARSession` object. To run `ARSession`, an `ARConfiguration` or its subclass is needed to determine how the device's position and motion is tracked.

ARConfiguration

`ARConfiguration`[3] is an abstract class that's used to set properties of the tracking which the session will be using. The set properties will be passed to session's `run(_ :options:)`.

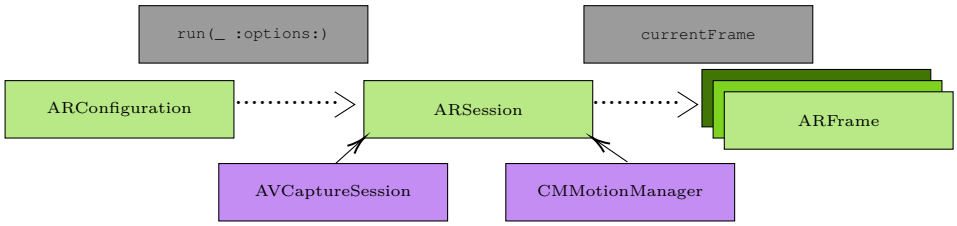

Figure 3.2: Structure of ARKit framework in context of application.

[2] https://developer.apple.com/documentation/arkit/arsession/
[3] https://developer.apple.com/documentation/arkit/arconfiguration/

An instance of `ARConfiguration` starts to run based on given configuration that immediately starts to collect the data from `AVCaptureSession` and `CMMotionManager`. This returns current frame, an instance of `ARFrame` class. This class represents all the information that application needs to render a virtual objects. The relationship is shown in the Figure 3.2.

ARKit includes following concrete configuration classes:

- **ARWorldTrackingConfiguration** — uses the back-facing camera to track device's orientation and position using 6DOF, rotation axes: pitch, roll, and yaw; and translation axes: x, y, and z. This configuration also supports methods for detecting planes, images, and objects added to scene as well as a method for hit-testing.
- **AROrientationTrackingConfiguration** — uses the back-facing camera that only works with 3DOF, rotation axes: pitch, roll and yaw. When using this configuration user can have an illusion of having 3-D object in real world as long as he pivots the device. If the user changes his position, the illusion is gone. Additionally, this configurations doesn't allow plane detection or hit-testing.
- **ARImageTrackingConfiguration** — uses the back-facing camera to only track the motion of reference 2D images in real-world. Image tracking is used 6DOF. This is same as `ARWorldTrackingConfiguration` but each configuration has its strengths. `ARImageTrackingConfiguration` requires lower performance and is better in moving situation, when the device is shaking. On the other hand, `ARWorldTrackingConfiguration` works better in stable, non moving environment.
- **ARFaceTrackingConfiguration** — uses the front-facing camera to detect face's position and orientation, its topology, and features that describe facial expressions. This information is provided by `ARFaceAnchor`.
- **ARObjectScanningConfiguration** — uses the back-facing camera to collect high-fidelity 3D scan of objects for later detection. This configuration is used only for development purposes as it has high performance and energy cost and disables ARKit features that aren't needed for reference scanning.

ARCamera

`ARCamera`[4] contains data about current camera position and characteristics about the captured video frame. One of the characteristics is tracking state. Since tracking is demanding process, its state can change during the AR session. To reflect these changes `ARCamera` has states that inform if the position tracking is, isn't available, or is limited. When the state is limited, `ARCamera` also has information about the most probable reason to let user know. These are initializing, relocalizing, excessive motion, and insufficient features.

ARFrame

`ARFrame`[5] contains a video image and position-tracking information that is being continuously captured AR session. This information is then delivered to other ARKit classes using `ARFrame`. There are two ways of receiving `ARFrame` objects:

[4] https://developer.apple.com/documentation/arkit/arcamera/
[5] https://developer.apple.com/documentation/arkit/arframe/

- Get the instance of `ARFrame` directly from `ARSession`.
- Make one of the objects delegate from `ARSessions` and let the object automatically receive new information as the device captures new frames.

ARAnchor

`ARAnchor`[6] holds information about position and rotation in real-world. This is basically 6DOF. `ARAnchors` can be dynamically added, removed, or modified during AR session. ARKit can using `ARConfiguration` automatically add anchors when they are detected and `ARConfiguration` supports them. Information about anchors that are in the scene can be obtained via instance of `ARFrame` or through the `ARSessionDelegate` that can update about creation or destruction of anchors. There are currently four types of `ARAnchors` that are supported by world tracking:

- `ARPlanceAnchor` has information about each flat surface that ARKit detects.
- `ARObjectAnchor` has information about 3-D objects that are detected and were specified by `ARReferenceObject`.
- `ARImageAnchor` has information about 2-D images that are detected and were specified by `ARReferenceImage`.
- `ARFaceAnchor` has information about pose, topology and expression of a detected face. When `ARFaceAnchor` is detected, it gives specified world alignment property to the face. This gives the face its own coordinate system which is then used to track if for example, the person is looking to the left or right. Blend shape property of `ARAnchor` gives the ability to detect facial expressions in terms of the movement of specific facial feature.

3.2 SceneKit

SceneKit[7] is framework that allows rendering of 3-D content using high-level scene descriptions. This means that there's no need to implement rendering algorithms. Unlike Metal or OpenGL, which is low-level and can also be used with ARKit.

SCNScene

`SCNScene`[8] is a class that is used to visualize 3-D content. It is containing a hierarchy of `SCNNodes` that are representing visual elements. Scene is usually created using Scene Editor built into Xcode.

SceneKit supported formats

SceneKit can read scene contents from a file in a supported format, or from an `NSData` object holding the contents of such a file. Supported formats include the following:

[6] https://developer.apple.com/documentation/arkit/aranchor/
[7] https://developer.apple.com/documentation/scenekit/
[8] https://developer.apple.com/documentation/scenekit/scnscene/

Format	Filename Extension	Supported in
Digital Asset Exchange	.dae	macOS 10.8 and later
Alembic	.abc	macOS 10.10 and later
SceneKit compressed scene	.dae or .abc	macOS 10.10 and later
SceneKit archive	.scn	macOS 10.10 and later

When a scene file in DAE or Alembic format is included into an Xcode project, Xcode automatically converts the file to SceneKit's compressed scene format for use in the built app. The compressed file retains its original `.dae` or `.abc` extension[9].

ABC file extension

ABC is file extension of the file used in Alembic[10] framework developed by Sony Pictures Imageworks. Alembic is data representation scheme for storing computer graphics scenes that focuses on efficiency of the results of complex procedural geometric constructions. This is practical for groups of people who are working on the same project. Alembic is not concerned with storing complex dependency graph of procedural tools but instead stores „baked results. This framework has been primarily used in visual effects and animation industry.

DAE file extension

DAE file extension is derived from "Digital Asset Exchange„. Files in DAE format identify COLLADA interchange file format for interactive 3-D applications. COLLADA is derived from "Collaborative Design Activity„.

This format represents a 3-D object or collection of objects as a surface formed by a collection of primitives, including triangles, polygons, and spline curves. They can also represent a full scene, including lighting, animation, and camera information. Optionally, vertex normals, colors, textures, and generalized shaders can be also included [22].

All scene content—nodes, geometries and their materials, lights, cameras, and related objects is organized in a node hierarchy with a single common root node. That's why COLLADA file format is the closest to the SCNScene (`.scn`) file format.

Files with dae extension can be opened as XML[11] files with various elements representing hierarchy of the 3-D objects in the librarial hierarchy. COLLADA XML language encodes both the content and the information on how to use the content (semantics). After the mandatory xml tag containing version and encoding of the document, there is mandatory COLLADA tag containing compulsory xmlns link to the COLLADA schema and version.

The first child element is `<asset>` element which must occur one time. Assets have two mandatory attributes: the creation date and last modification. `<asset>` element is becoming more important as it contains child elements with data about authors, contributors date and time of creation and many more with new elements being added in new versions. However, the basic needs are nearly completed. These problems were created mostly due to the copyright issues in entertainment industry.

Next, the elements represent individual parts of the 3-D object. Nevertheless, as the data sets become larger and more complex, they become harder to manipulate withing a single element. Common approach to manage this complexity is to divide the data into smaller pieces organized by some criteria. These modular pieces can then be stored in

[9] https://developer.apple.com/documentation/scenekit/scnscenesource/
[10] https://www.alembic.io/
[11] https://en.wikipedia.org/wiki/XML

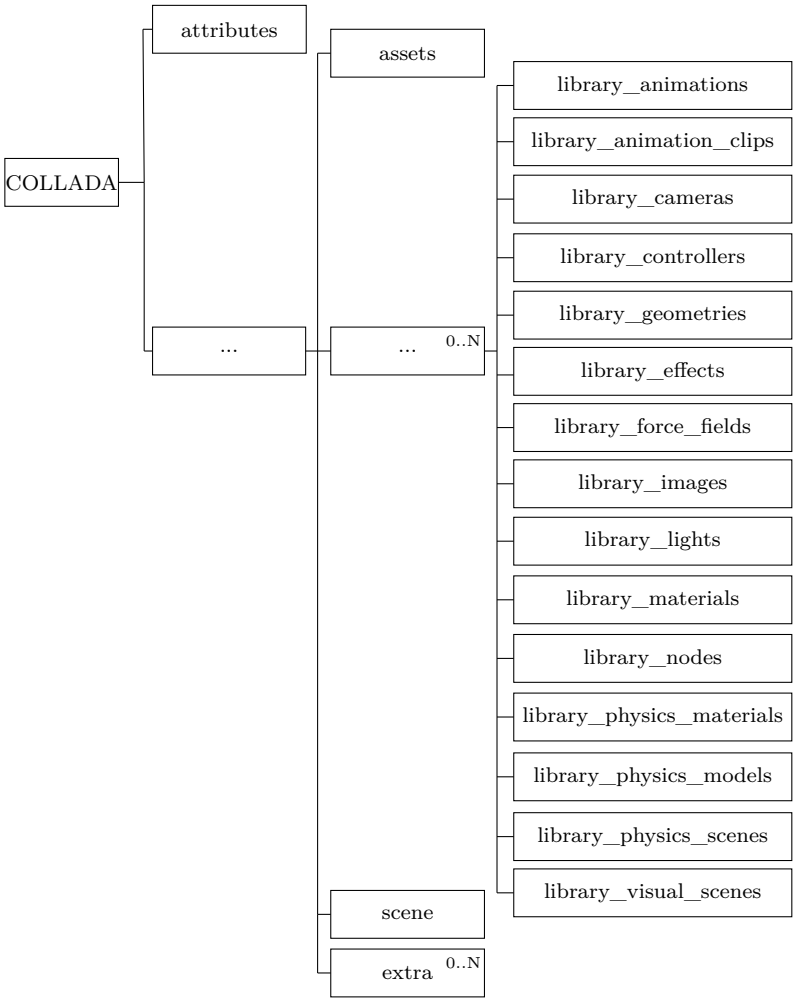

Figure 3.3: Structure of COLLADA file in XML. [3]

separate resources as libraries. These elements can have one optional `<asset>` child and zero or more `<extra>` elements. Other than that, libraries contain specific child elements of their own type. For example, `<library_animations>` element can contain multiple <animation> child elements for each animation of the object. This is useful for representing the movement of several parts of a more complex object, i.e. walking consists of moving all the body parts. The library elements cover all the necessary aggregations of the 3-D object (as seen in Figure 3.3).

An optional `<scene>` element describes the entire set of information that can be visualized from the contents of a COLLADA resource. After the `<scene>` element there might be zero to maximum occurrences of the `<extra>` element that is unbounded. `<extra>` element can be also used inside of the children of library elements. Since COLLADA 1.0, and on was the content of `<extra>` element changing. At first, it was used to store arbitrary text data. In COLLADA 1.1, the `<extra>` element was a container of `<param>` elements. In the newest COLLADA 1.5, the `<extra>` element represents an asset of well-formed XML content contained in `<technique>` element. This means that the `<extra>` element accommodates any type of information in categories according to technique and profile, in

other words, additional user-defined or application-specific information. Information of the `<extra>` element can be also managed as an asset.

SCNView

`SCNView`[12] is subclass of UIView that's used for displaying 3-D content. As subclass of `UIView`, it is part of user interface. This means that if `SCNView`'s scene property has assigned scene, developer can display the scene by adding `SCNView` to the application's user interface. `SCNView` let's developers configure the properties of a view. These can have significant impact on device's performance as you can choose the preferred frames per seconds (FPS) which changes the rendering frequency of the scene. `rendersContinously` changes whether or not the the scene is always rendered at its preferred FPS or only when the content changes. The last property is antialiasing mode. This property can set in which mode are the jagged edges reduced into looking smoother. `SCNView` also allows to change camera control configuration properties.

ARSCNView

This class is derived from `SCNView` and connects ARKit with SceneKit by adding real-world background video from `ARFrame`. `ARSCNView` automatically translates the coordinates between real-world and virtual scene. On top of that, `ARSCNView` also updates the position `SCNCamera` as the device's moves in the real-world.

With every detected `ARAnchor` that ARKit reads, `ARSCNView` creates `SCNNode`. These nodes can be used to attach 3-D virtual content into the AR session. To get the information about changes of these `SCNNode`, `ARSCNViewDelegate` is used.

SCNNode

`SCNNode`[13] is an element representing a position and transform in a 3-D space. `SCNNode` only represents coordinate transform relative to its parent.

The main node is `rootNode` which defines the coordinate system of the world (see Figure 3.4). Each child attached to `rootNode` creates its own coordinate system which is relative to node's parent node. A node can be transformed by editing its position, rotation and scale properties or using transform property. The node hierarchy determines spatial and logical structure of the scene. However, determining whether node is 3D object, light or camera is done by attaching objects to nodes. There are three main types of nodes in SceneKit: `groundNode`, `lightNode`, `cameraNode`. Without these objects, `SCNNode` has no visible content. Each object has different attributes that can specify its function. For example, camera has camera perspective, field of view, visual effects and more.

There are different types of `lightNodes` in the SceneKit as well as in the 3-D graphics. The most general are:

- **Ambient** — the light hits the object equal amount from all the directions. This means that there are no shadows.
- **Directional** — the light has a direction but no source.

[12] https://developer.apple.com/documentation/scenekit/scnview/
[13] https://developer.apple.com/documentation/scenekit/scnnode/

- **Omni** — the light has a direction (which is similar to directional). but also a position. This can be useful in calculations of light intensity in given distance.
- **Spot** — the light has a direction and a position (similar to omni), but it falls off in intensity in a cone shape.

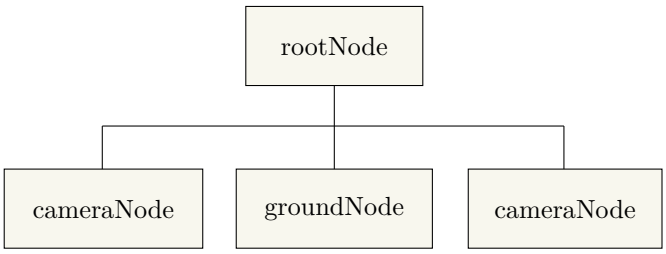

Figure 3.4: Datagram showing node hierarchy in SceneKit.

SCNAction

`SCNAction`[14] is simple and reusable animation that changes attributes of the attached node. Actions provide a way to implement animated behaviours of the `SCNNode` object. Examples of the actions are: move, rotate, scale, fade, hide, or play audio. These animations provide change of the properties in the node during the time period longer than one frame. Duration property is used to properly set the time that the it takes to complete the action. `TimingMode` property defines the rate at which the animation executes. To adjust the timing by increasing or decreasing playback speed, a speed property is used. Using action's `reversed()` method, animations can be reverted to their previous state. Reverting animations are frequently used immediately after the initial action. To group or stack the actions into one parental action, `SCNAction` has the possibility to combine actions together and create more complex behaviors of the nodes:

- **Sequence action** — each action in the sequence follows the previous action as soon as it ends.
- **Group action** — all of the actions stored in the group are played at the same time.
- **Repeating action** — repeating action contains only one action. When the child is completed, it starts to play over again.

SCNAnimationPlayer

`SCNAnimationPlayer`[15] is a class introduced in iOS 11 that adds the control to the animation even during the animation. It has options to start, stop, or even change the speed of an animation.

[14] https://developer.apple.com/documentation/scenekit/scnaction/
[15] https://developer.apple.com/documentation/scenekit/scnanimationplayer/

SCNTransaction

`SCNTransaction`[16] can be also used for more complex animations. Methods of `SCNTransaction` class animate the changes of animatable properties of the scene and combines them into nested transactions. SceneKit automatically creates transactions by modifying properties of the scene. Those are automatically applied at the end of the of the loop in the current thread that the changes are made in. The modifications will happen instantly, because the animation duration of the transactions has default value of zero. Properties such as animation duration are modified using `SCNTransaction` methods.

CAAnimation

CAAnimation[17] is the abstract superclass for animations in Core animation. This class is not instanced, instead specific subclasses are instanced: CABasicAnimation, CAKeyframeAnimation, CAAnimationGroup, or CATransition.

Core animations can be used in SceneKit by attaching CAAnimation to the SceneKit object like node. This way SceneKit can represent animations created by external 3-D tools. To attach the animation into the object, SCNAnimatable (see section 3.2) protocol is used.

SCNAnimatable

SCNAnimatable[18] is a common interface for attaching animations to the SceneKit objects such as nodes, geometries, and materials. Since SceneKit uses the same architecture as the Core Animation framework, animations can be created both implicitly and explicitly. Implicitly by using `SCNTransaction` and explicitly by `CAAnimation`. This interface can be also used to control any of the animations that are already attached to a SceneKit object.

Physics Simulation

For interactions between object, SceneKit has classes that enable physics simulation. There are two classes that enable interaction to node.

Firstly, `SCNPhysicsBody` adds simulation attributes to the `SCNNode`. These attributes define if the object can react to the impulses from other objects, if it can apply its own force onto another object. The physical body properties such as mass, charge, or friction could also be changed. These properties can make each object react uniquely.

Secondly, `SCNPhysicsShape` creates shape that is interacting in physics simulation. There are two ways of creating it. Either by rendered geometry of the object or by `SCNNodes`. `SCNPhysicsShape` has also ability to create new shape by combining other shapes.

3.3 ARCore

ARCore[19] is platform for building AR experiences developed by Google. ARCore provides APIs for essential features of AR. Three key technologies are:

- **Motion tracking** used to track the phone's position relative to the real environment.

[16] https://developer.apple.com/documentation/scenekit/scntransaction/
[17] https://developer.apple.com/documentation/quartzcore/caanimation/
[18] https://developer.apple.com/documentation/scenekit/scnanimatable/
[19] https://developers.google.com/ar/

- **Environmental understanding** allows the device to detect the size and location of flat horizontal, vertical and angled surfaces such as walls, ground, or desk.
- **Light estimation** is used to detect the current lighting conditions of the real-world environment.

ARCore supports both Android and iOS. In fact, the list is even longer and supports most of the developers:

- Android
- Android NDK
- Unity for Android
- Unity for iOS
- iOS
- Unreal

3.4 Comparision between ARCore and ARKit

Both of these frameworks do a great job of visualizing virtual objects into a real-world scenes. Even though the differences between those two platforms are subtle, they distinguish each framework and help with decision making.

When it comes to compatibility and how wide-spread the frameworks are, ARKit can run only on devices with iOS - iPhones, iPads. On the other hand, ARCore is targeted on multiple device manufacturers because it runs primarly on android. Even though 85% of the world uses android, there are still more users with ARKit supported devices than ARCore supported devices. This is because ARCore is only supported on very few flagship android phones or has software limitation. To put this in numbers, according to this article by 9to5Google [13], in the December of 2018, there were 250 million ARCore compatible devices. This makes it only about one fourth of total 990 million ARKit and ARCore supported devices combines calculated by AR Insider [13] during the same time.

Due to different hardware, ARCore applications often show signs of more positional lag than the ones using ARKit. The reasoning might not be just different sensors calibrated in different ways but just something to be fixed in next versions.

To conclude, the most notable advatage of ARCore is in mapping. ARCore can hold larger map data leading to more stable data set. On the flip side, ARKit only utilizes a “sliding window„ storing only limited amount of previous data but offers much better tracking reliability. ARKit also relies only on already-installed technology while ARCore needs to have dependencies installed.

Chapter 4

Design of the solution

The problem is when the user wants to display animated 3-D virtual objects that might be already created in a format that isn't supported by SceneKit. Animated means that the object can either have animation or can change its state. Light switch that has two positions - on and off, is great example of simple animated object. User can interact with the object and make it change its state. The change of a state is displayed as an animation.

The main goal of this thesis is to allow user to visualize such objects in augmented reality on iOS platform using ARKit. After displaying the object, user can interact with it. Interaction means to change object's state, position or scale the object. It is important for user to have good user interface that allows him to have all the important controls, yet it still leaves space for the visualization in AR.

4.1 Application requirements

Requirements for the application consist of making sure that the user can both interact with the application well, that is part of good user interface as well as meeting all the uses that the application might be used for.

Application has simple user interface that provides all the functionality that user needs:

- Plane detection that scans for usable places to visualise the item displayed by yellow square in the middle of the screen that blinks when a usable plane is detected.
- A "plus„ button that let's used add object if it's possible to add an object to the scene. After tapping the button, context menu shows up with all the available objects. By tapping the object icon, objects appears in the scene.
- A "trash bin„ button that changes state when tapped let's user delete the object from scene by tapping the trash bin button and then tapping the object which the user wants to delete.
- A rotate gesture that let's user rotate selected object by rotating the fingers on the screen in circular motion.
- Pinching the object in the scene out or in scales the object.
- Tapping on the object animates the object.

As far as the user interaction goes, user should be able to place an object into the scene if it's possible in the current circumstances. After placing the object in the scene, user shall

be able to move the object in the scene as well as modifying some of it's properties such as scale, rotation, in some cases even the materials of the object. User should be able to also delete the object if he decides to.

4.2 Potential use-cases

Using my application helps the user to imagine a product in specific environment. This makes for new user experience which can be more compelling when presenting the product.

- User can visualise the product that can be customized in the specific place. Based on the visuals he then can make better decision if the product suits the space. If not, he can decide if changing the scale, position or even the color would help. This gets user feedback which leads to more information about the specifics of the customization. Furthermore, this can have psychological effect on the user when he sees the product in his own space.
- Another use-case is to present a prototype product without the need to create a physical model. Presenter can create more models with slight alternations that would be costly otherwise. People tend to find possible future problems sooner if they are presented with prototype. An actual product can be the produced based on the gathered information.

4.3 User interface

A button with plus logo was used to choose the object which should be added to the scene. Position of the button is in the middle bottom. This way the user has the most screen real-estate available for the AR experience. After tapping the button a list of available objects is shown in which user can navigate through the objects by swiping. Since focus square is used to detect the plane, the object is is automatically added after tapping on it.

Deletion of an object is done by a button in bottom right corner of the screen. To delete an item, user has to tap the trash bin icon. As a notifier, that the user is in the deletion mode, trash bin icon has it's lid opened. When in deletion mode, user selects which object does he want to delete by tapping on it. Using hit testing, results are inspected for any nodes that they may contain. If any of the result belong to the virtual object, object is deleted.

Since these functions are frequently used by the user, they should be always available for the user. That's why these buttons are on the bottom of the screen, so that user doesn't have to reach for them.

Other functions as resizing, moving and animating the objects are done by common gestures over the image so that they don't block the view and make the AR experience more immersive.

Figure 4.1: Layout of the buttons at the bottom of the device's screen.

4.4 File formats

One of the main goals of the thesis is to display the objects that were already created. This usually means objects that don't have the needed format to work with SceneKit and ARKit. As mentioned before, supported formats are Alembic and COLLADA. In order to display other file formats, they need to be converted into one of the two file formats. Because different formats use different methods of representing the objects and every 3-D object represents different real-world object, there isn't complex way to create unified files to use in the application automatically.

Problems with converting files

Some of the problems that arise when I tried to just converting the files without modifying some of the properties were that the object in the scene wasn't placed in the origin of the scene coordinates, so the object appeared as if it was in the distance. If displayed in the application, this problem lasts even if the object is moved in the scene as the user controls the origin position. While this problem can be fixed in the Xcode, it is not always easy to find the origin and also the object at the same time, because the editor is limited. This is especially the case when the object is far away.

In a scenario, when the object is scaled bigger than wanted user can't change the size of the object in the Xcode, only the scale. If the displayed object is too big in the application, user doesn't have to option to scale to object down, as he appears to not see any objects because he's "inside„ the object.

Prepartion of the objects

The solution to this is to manually modify some of the properties in a 3-D graphics software toolset of choice. I use Blender[1] as it's open-source and provides all the needed functionality.

After importing the file into the software, it is important to set the the correct origin of the 3-D object. Next step would be to scale the object appropriately to it's real-world form. In case of animated objects, good practice is to run the animation and check if the animation doesn't have a blank space, meaning no animation, towards the end of the timer. If it has an unwanted space, shortening the animation as it can't be done in the Xcode.

When all the modifications are made, exported the file as COLLADA file with `.dae` extension or in Alembic file format with `.abc` extension (Alembic can't be used to export explicit animations to the Xcode) is run through the script `stripForXcode.py` based on the script by Jon Cardasis[2] that deletes unusable `<library_geometries>`, `<library_materials>`, and `<library_images>` elements that can't be used in the Xcode.

Final object file imported into the Xcode must be converted to the `.scn` which is optimized for the use in the SceneKit.

Lights

There are more types of light that can be used in the SceneKit (see section 3.2). As far as types of lights go in this application, omni light is used to the visualised 3-D object in the scene. This type of light make the object look more realistic by changing its light intensity

[1] https://www.blender.org/

[2] https://gist.github.com/joncardasis/e815ec69f81ed767389aa7a878f3deb6/

and light temperature according to the surrounding of the object in the current `ARFrame`. This estimate is calculated in renderer function and then updated into the scene light.

4.5 Animating the objects

There are two main ways of animating objects in SceneKit. Using explicit (COLLADA) files and translating individual scene nodes implicitly.

Animation using COLLADA files

Firstly, there is a way to animate objects using biovision hierarchical data file format or BVH [8] that stores motion data alongside skeleton hierarchy of an 3-D object. This is possible, because SceneKit's animation have the same structure to the Core Animations. During the research, I found that the most convenient API for loading the explicit - COLLADA animations is SCNAnimationPlayer introduced in 2017[3] as it uses SceneKit. This API replaced the way of animating SceneKit objects with `CAAnimation` function `animation(forKey:)` that was deprecated.

COLLADA animations can be added or configured to the 3-D model which can have a bone structure in most of 3-D creation software programs such as Blender explicitly. In case of my project, they need to be exported as COLLADA files with `.dae` extension. Models that use Alembic `.abc` extension don't show animations in the Xcode, instead they are displayed the way as `.scn` - without animations. Alembic files also don't have an option in the Xcode to be converted into files with `.scn` extension. Inside the COLLADA files, the animations are stored into `<library_animations>` element in the XML representation of the file.

Xcode requirements

The need of COLLADA file is in spite of Xcode. Apple supports animations with one `<library_animations>` element that can have multiple animation element as children. The exact form of the animation wasn't required in previous versions of Xcode, however it was introduced with the release of macOS Sierra and Xcode 9. Multiple `<animation>` elements are commonly used to move different parts of the 3-D model separately. Nevertheless, this approach doesn't work in the Xcode properly. To get animations to work as expected, there should be only one animation children tag inside of the COLLADA file. This means that each of the `<library_animations>` element's children need to be morphed into one in order to be displayed in Xcode correctly. Such model could have <animation> elements for movement of each limb during more complex animation like running. To make an animation display in Xcode, it needs to be put into leading and trailing `<library_animations>` elements with only single `<animation>` element between them. After morphing the multiple `<animation>` elements into one, a `<source>` element is used to differentiate between the elements that were previously `<animation>`.

Transforming to Xcode

In order to have all the COLLADA animations in correct format for the Xcode, a python script `stripForXcode.py` was created that deletes all the unnecessary `<animation>` ele-

[3] https://developer.apple.com/videos/play/wwdc2017/604/?time=2036

ments and leaves only one. The output of the script are three files. First file is the original unchanged file with .old extension used as a backup. Other two files are modifications of the original file. One contains all the original elements excluding only `<library_animation>` element and it's children. This file is later used as a model that will be displayed in the application. It can be also converted into `.scn` file extension in the Xcode. The last file consists of all the libraries except geometries, materials and images. Necessary libraries that are left are the animations and bone structure of the object. Animations are transformed into single `<animation>` element. This file will be later used as the animation attached to the 3-D object in the scene. To differentiate between the files, one with the animation as "anim-„ prefix.

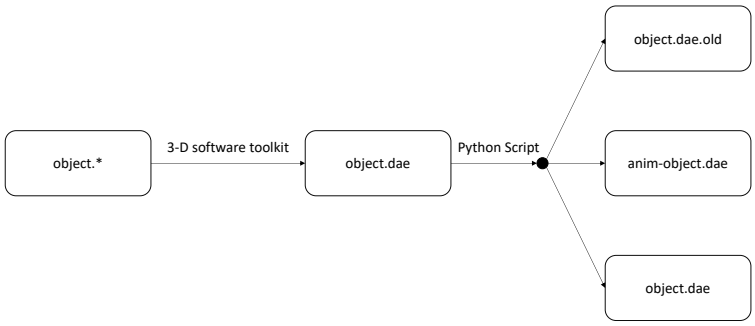

Figure 4.2: Diagram showing the preparation of a file.

Animation by transforming nodes

Secondly, to implicitly animate 3-D objects in SceneKit `SCNAction` and `SCNTransaction` classes can be used. This approach relies on the knowledge about the object in the scene. It is important to know at least the animatable node structure of the object or move only the root node. Animating objects this way is easier with simple objects that have one or very little nodes. Such animations can be done using hit testing for any node and transforming the the very first or all of the nodes hit. In case of more detailed object, each node may have different purpose and couldn't be transformed in the same way as the other one can. Because of this, it is necessary to know how to differentiate between the parts of the scene when modifying animatable parts using `SCNAction` and `SCNTransaction`.

This form of animating was used in a demo shown at Excel@FIT[4] that was part of an application for multipeer connectivity. It has more potential to be used in a commercial sphere since it typically uses only elementary animations. Example of such is wardrobe, that can be opened and closed anywhere in the home to pursue the customers.

[4] http://excel.fit.vutbr.cz/

4.6 Knight application

As a part of Excel@FIT conference, I created an application together with Adam Jurczyk to demonstrate the usability of the ARKit with use of Multipeer Connectivity[5]. The aim of the application is to showcase a possible solution that allows multiple people to interact with a single customizable object in a shared augmented reality session. It also demonstrates 3D object handling and customization with changes synchronized in real-time to all connected users, as well as resolving conflicts resulting from multiple people interacting with the same single object.

MultipeerConnectivity is a native Apple framework that handles communication between devices using underlying networks. It can use infrastructure Wi-Fi, Bluetooth or direct Wi-Fi.

I've provided models and methods for displaying the models, interacting with the model. Interacting means that the model can be placed in the space, animated, drag onto the objects (planes), resized, and rotated. Additionally, the model has the ability to change colors of the parts of the body using buttons on the side of the workspace. All of these interactions are translated into every user's phone at the same time. When the user interacts with one part of the body, the part is being highlighted and locked, so that other people can't manipulate with the same part of the body. However, users can change other properties of the knight.

This implementation was demonstrational and it's goal was to showcase the possibilities of the displaying 3-D models in AR on multiple devices. Application with the same structure, only different models can be used for marketing purposes. Because of the knowledge about the nodes in the model, I was able to implement changes of the properties. This meant changing to color or the nodes. That's why animating using `SCNAction` was used in this case. It is simple and can be used in stateful models.

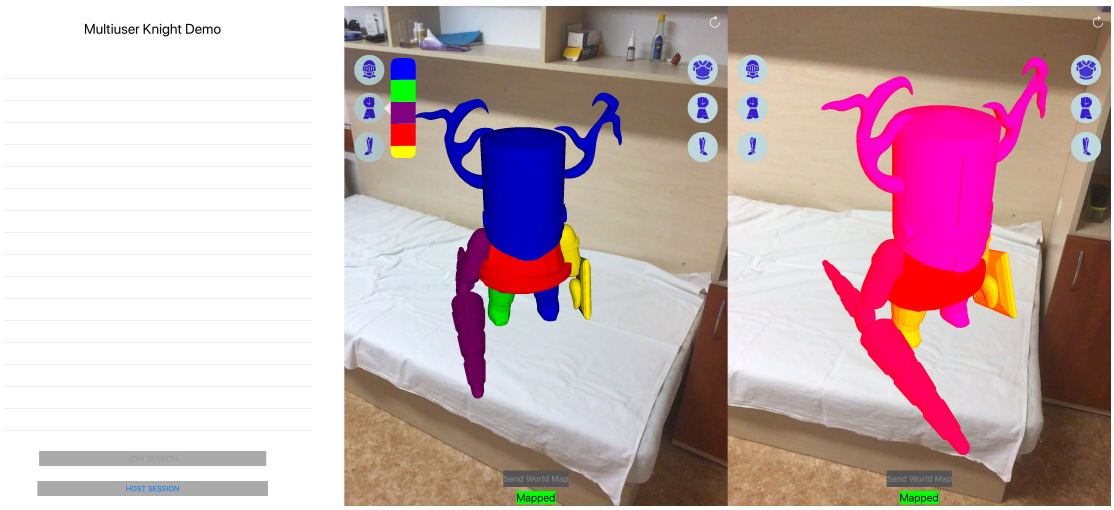

Figure 4.3: Screenshots of Knight application.

[5] https://developer.apple.com/documentation/multipeerconnectivity

Chapter 5

Implementation

The aim of the application is to display animated 3-D objects in augmented reality on iOS. With this in mind, I chose to develop the application using ARKit framework for working with augmented reality on iOS devices. Intended version for running the application is iOS version 12 but app is supported in iOS 11 and higher. The application is developed on Apple iPhone 6S and Apple iPad 6 running the latest iOS 12.3. Chosen framework used for rendering the 3-D content in augmented reality is SceneKit. SceneKit suits the use-case of the application the best, as it offers full integration with ARKit and is intended for use like displaying the objects without the need of special effects or demanding animation.

Final application shouldn't be standalone application that has ability to convert object from any format and display it in 3-D but has the ability to be extended to such width in the future. However, the process of converting to the SceneKit compatible format is explained later on.

5.1 Application structure

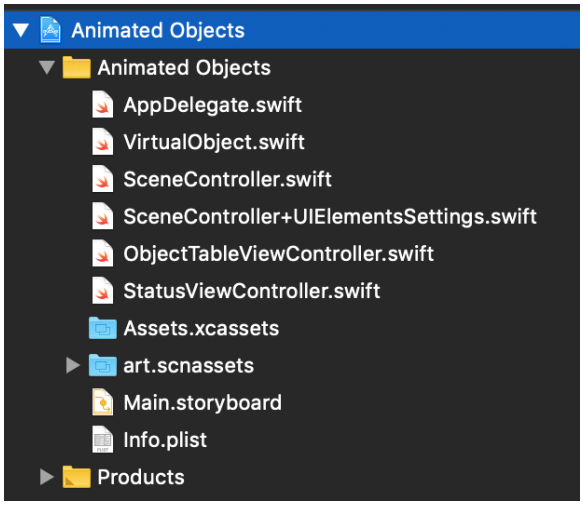

Figure 5.1: Structure of the files in the application displayed in the Xcode.

Source code of the application has following structure:

- **VirtualObject.swift** — class file that contains information about virtual object in the scene. Some of the notable properties are position, rotation, scale, geometry, but also information about the animation that belongs to the object
- **SceneController.swift** — main class file that controls the `SCNScene`, `ARSession`, takes care of adding and deletion of the objects and movement with the objects
- **SceneController+UIElementsSettings.swift** — controller file that controls the segue into table view with all the objects
- **ObjectTableViewController.swift** — controller file that controls the interaction in table view and cell as well as delegating scene controller
- **StatusViewController.swift** — controller file that controls the state of tracking the environment
- **Assets.xcassets** — application icons
- **art.scnassets** — 3-D animated objects, textures
- **Main.storyboard** — launch screen with buttons and `ARSCNView` scene
- **Info.plist** — information property list about the application

5.2 Scene configuration

Application uses `ARWorldTrackingConfiguration` so that plane, object detection and other features are available. This means that the device needs to support `ARWorldTrackingConfiguration`, otherwise the application wouldn't run. In the `Info.plist` file of the application is "arkit„ under "Required device capabilities„. This only means that the application will be displayed in the App Store only to the people who meet the requirements. However, user may still be able to install it, therefore it's being checked. Checking whether or not the device is supported is done in `viewWillAppear()` function.

5.3 Displaying the object

The object is represented by `VirtualObject` class that is derived from `SCNReferenceNode` class. It is used as a placeholder for the content loaded from assets. `VirtualObject` provides extra class variables holding information about the object like if there is an animation available, animation and play state of the animation.

Loading objects

Loading all the objects from `art.scnassets` is done by static function `getVirtualObjectFromNode(_ node:)` of the class `VirtualObject`. This function iterates through all of the `.scn` files and returns them as an array of `VirtualObjects` to be used in the object listing. Through this function gets information about all the models that can be used in the application when added to the `art.scnassets`.

Displaying objects

The fact, that `VirtualObject` class inherits from the `SCNNode` makes adding objects to the scene easier. When an object is selected, a corresponding anchor is created in the `sceneView`. Attaching `VirtualObject` to the `ARAnchor` is done in renderer method. To display content of an object, `load()` method must be called to add all children of the scene file's root node as a children of the VirtualObject.

Light

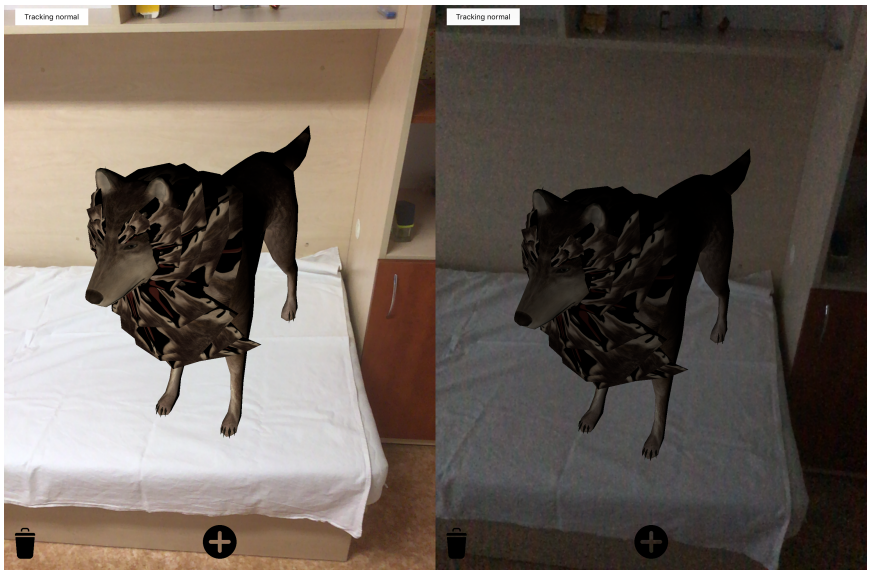

Figure 5.2: Difference in the light intensity and light temperature between two objects in different light conditions.

To make the objects looks more realistic, I implemented Physically Based Rendering to estimate the lighting in the scene. The estimate is calculated in renderer function and then updated into the scene light in the time frames animating the light change to make it more seamless.

```
1 func renderer(_ renderer: SCNSceneRenderer, updateAtTime time: TimeInterval
      ) {
2         // Get light estimate from current frame
3         guard let lightEstimate = self.sceneView.session.currentFrame?.
              lightEstimate else { return }
4         let ambientLightEstimate = lightEstimate.ambientIntensity
5         let ambientColourTemperature = lightEstimate.ambientColorTemperature
6         // Change light parameters in the scene accordingly
7         sceneLight.intensity = ambientLightEstimate
8         sceneLight.temperature = ambientColourTemperature
9     }
```

Listing 5.1: Estimattion of the light.

5.4 Interaction with the objects

To detect the objects in virtual environment a method hit `hitTest(_ point: options:)` is called every time any of the gesture recognizer methods is called. By comparing the results, the gesture recognizer functions decides whether or not to move to object.

Positioning the object

Moving the object is done by simple function that responds to pan gesture. If the gesture started at the object node, this method then repositions all of the object's nodes each frame to the position of users of user's finger if there is a valid feature point. This approach gives more freedom in object positioning than the pane recognition.

```
1 guard let hitTest = sceneView.hitTest(gesture.location(in: self.sceneView),
      types: .featurePoint).last else { return }
2 let worldTransform = hitTest.worldTransform
3 let newPosition = SCNVector3(worldTransform.columns.3.x, worldTransform.
     columns.3.y, worldTransform.columns.3.z)
4 objectToMove.simdPosition = float3(newPosition.x, newPosition.y,
     newPosition.z)
```

Listing 5.2: Positioning the objects accoring to the world coordinates (simplified).

Rotating the object

Rotating the object is implemented by rotation gesture handler that relies on class variable `currentAngleY` from `VirtualObject` that stores the information about how much should the angle change from the current position.

```
1 var currentAngleY: Float = 0.0 // in VirtualObject class
2 ...
3 let rotation = Float(gesture.rotation)
4 if (gesture.state == .changed) {
5     objectToRotate.eulerAngles.y = objectToRotate.currentAngleY + rotation
6 }
7 if (gesture.state == .ended) {
8     objectToRotate.currentAngleY = objectToRotate.eulerAngles.y
9 }
```

Listing 5.3: Rotation of the objects using Euler angles (simplified).

Scaling the object

Scaling is done by pinch gesture followed by multiplying of the object's scale value in `CGFloat`.

```
1 let pinchScaleX: CGFloat = gesture.scale * CGFloat((objectToScale.scale.x))
2 let pinchScaleY: CGFloat = gesture.scale * CGFloat((objectToScale.scale.y))
3 let pinchScaleZ: CGFloat = gesture.scale * CGFloat((objectToScale.scale.z))
4 objectToScale.scale = SCNVector3Make(Float(pinchScaleX), Float(pinchScaleY)
     , Float(pinchScaleZ))
```

```
5  gesture.scale = 1
```

Listing 5.4: Scaling the object using pitch gesture (simplified).

5.5 Animation

```
1  extension SCNAnimationPlayer {
2      class func loadAnimation(fromSceneNamed sceneName: String) ->
           SCNAnimationPlayer {
3          let scene = SCNScene( named: sceneName )!
4          // Find the top level animation
5          var animationPlayer: SCNAnimationPlayer! = nil
6          scene.rootNode.enumerateChildNodes { (child, stop) in
7              if !child.animationKeys.isEmpty {
8                  animationPlayer = child.animationPlayer(forKey: child.
                       animationKeys[0])
9                  stop.pointee = true
10             }
11         }
12         return animationPlayer
13     }
14 }
```

Listing 5.5: Core Animation.

An extension of `SCNAnimationPlayer` is used to to run the animation within COLLADA file. This extension has a method that scans the COLLADA file for any usable animations and returns just the animation. Located animation is then attached to the object.

Figure 5.3: Object during animation.

Loading the application is done by `SCNAnimationPlayer` API. Once an instance of the animation is created, animation can be attached to the object. After attaching the animation to the object, it is automatically playing. This is unwanted, so the `stop()` function is called.

```
1 let animation = SCNAnimationPlayer.loadAnimation(fromSceneNamed: "art.
     scnassets/anim-sample.dae")!
2 sampleObject.addAnimationPlayer(animation, forKey: "animation")
```

Listing 5.6: Initializing animation and attaching it to the scene.

5.6 Testing

Testing of the application was done on a small group of approximately five people multiple times during the implementation of the application. Apart from that, I had an opportunity to get feedback from more that 15 people that I showed the demo to at Excel@FIT.

The most important factor in testing was usability and user experience. When asked how to add or delete the object from the scene, all of the people figured a way to do it without any help. Even though the task itself wasn't challenge, some people were confused when they had to scan the scene prior to the insertion. After the first test I ordered them to change the scale, move the object and rotate. Scaling and moving the object was very intuitive for most of the participants, but rotating was polarizing. Some people preferred slider, other the gesture. Comparatively, more people leaned towards the gesture operating than the slider. Other than that, some minor changes were made to the multipeer session section.

Chapter 6

Conclusion

The main goal of this bachelor thesis was to create an application capable of displaying 3-D objects in augmented reality. Furthermore, to create an application that can interact with animated objects. It was necessary to design and implement a way of converting the objects and animations to the state in which they are usable in the application.

At first, it was necessary to get in touch with augmented reality concepts, all of the frameworks that could be used and distinguish the right ones to use. Following was the research of the file formats and the solution to the problem of both displaying the objects and animating them.

As a solution I designed a way of converting both the objects and animations to be in the compatible format by splitting them into two separate files and changing their XML structure. To display the objects I designed and implemented an application that loads the objects, let's user place them in the real-world environment, manipulate with them and run their animations.

As a part of an Excel@FIT conference, I participated in the creation of an Knight application for which I provided a way of displaying the objects, changing the properties of the objects and interaction with the objects.

While the application isn't in the state, where it's ready to be published and used in commercial way, there is big potential in it. By adding some more features such as synchronizing the objects with the application through cloud and adding multipeer session functionality to the system, it could provide unique and powerful solution in the marketing sphere.

Bibliography

[1] 3rockAR Team: Augmented Reality Is Transforming Museums. [online, 2019.04.13]. Retrieved from: https://www.3rockar.com/augmented-reality-transforming-museums/

[2] Apple Inc.: Detecting Images in an AR Experience. [online, 2019.04.13]. Retrieved from: https://developer.apple.com/documentation/arkit/detecting_images_in_an_ar_experience

[3] Arnaud, R.; Barnes, M. C.: *COLLADA: sailing the gulf of 3D digital content creation*. AK Peters/CRC Press. 2006.

[4] Audi, A.; Deseilligny, M.; Meynard, C.; et al.: Implementation of an IMU Aided Image Stacking Algorithm in a Digital Camera for Unmanned Aerial Vehicles. *Sensors*. vol. 17. 07 2017: page 1646. doi:10.3390/s17071646.

[5] Birkfellner, W.; Figl, M.; Huber, K.; et al.: A head-mounted operating binocular for augmented reality visualization in medicine - design and initial evaluation. *IEEE Transactions on Medical Imaging*. vol. 21, no. 8. Aug 2002: pp. 991–997. ISSN 0278-0062. doi:10.1109/TMI.2002.803099.

[6] Blakemore, E.: There's augmented reality in space now. [online, 2019.03.24]. Retrieved from: https://theweek.com/articles/607794/theres-augmented-reality-space-now

[7] Durrant-Whyte, H.; Bailey, T.: Simultaneous localization and mapping: part I. *IEEE robotics & automation magazine*. vol. 13, no. 2. 2006: pp. 99–110.

[8] Gleicher, M.: CS838 - Topics in Computer Animation. [online, 2019.02.04]. Retrieved from: http://research.cs.wisc.edu/graphics/Courses/cs-838-1999/Jeff/BVH.html

[9] Hyundai Motor Group: Hyundai & WayRay Unveil Next-generation Visual Technology at CES 2019. [online, 2019.04.13]. Retrieved from: https://www.hyundai.news/eu/technology/hyundai-wayray-unveil-next-generation-visual-technology-at-ces-2019/

[10] Iram, S.: Why Web Apps Are The Future Of Augmented Reality. [online, 2019.04.13]. Retrieved from: https://medium.com/arjs/why-web-apps-are-the-future-of-augmented-reality-c503e796a0c5

[11] Joseph, S.: Ikea wants to consolidate its three mobile apps into one. [online, 2019.04.13].

Retrieved from: https://digiday.com/marketing/ikea-wants-consolidate-three-mobile-apps-one/

[12] Journal of Mobile Technology in Medicine: Augmented Reality Surgery using iPads. [online, 2019.04.13].
Retrieved from: https://www.journalmtm.com/2013/augmented-reality-surgery-using-ipads/

[13] Li, A.: Google ARCore now supports 250 million devices as version 1.6 rolls out. [online, 2019.03.24].
Retrieved from: https://9to5google.com/2018/12/07/arcore-1-6-rolling-out/

[14] Livingston, M. A.; Rosenblum, L. J.; Brown, D. G.; et al.: *Military Applications of Augmented Reality*. chapter Military Applications of Augmented Reality. New York, NY: Springer New York. 2011. ISBN 978-1-4614-0064-6. pp. 671–706. doi:10.1007/978-1-4614-0064-6_31.
Retrieved from: https://doi.org/10.1007/978-1-4614-0064-6_31

[15] Nejati, M.; Nejati, M.: *Global Business and Management Research: An International Journal Vol.2, No. 2 & 3*. Universal Publishers. 2010. ISBN 9781599425818.
Retrieved from: https://books.google.cz/books?id=PkvjLMfrtrUC

[16] Rey, J. D.: Amazon's new 3-D feature is augmented reality that people might actually use. [online, 2019.02.04].
Retrieved from: https://www.recode.net/2017/11/1/16592238/amazon-app-augmented-reality-ar-view-3d

[17] Saidin, N. F.; Halim, N. D. A.; Yahaya, N.: A review of research on augmented reality in education: advantages and applications. *International education studies*. vol. 8, no. 13. 2015: pp. 1–8.

[18] Schmalstieg, D.; Hollerer, T.: *Augmented reality: principles and practice*. Addison-Wesley Professional. 2016.

[19] Stein, S.: Apple's multiplayer AR vision of the future works pretty well so far. [online, 2019.04.13].
Retrieved from: https://www.cnet.com/news/apples-multiplayer-arkit-2-vision-of-the-future-works-well-so-far/

[20] Sung, D.: Augmented reality in action - maintenance and repair. [online, 2019.04.13].
Retrieved from: https://www.pocket-lint.com/ar-vr/news/108887-augmented-reality-maintenance-and-repair

[21] Sutherland, I. E.: A Head-mounted Three Dimensional Display. In *Proceedings of the December 9-11, 1968, Fall Joint Computer Conference, Part I*. AFIPS '68 (Fall, part I). New York, NY, USA: ACM. 1968. pp. 757–764. doi:10.1145/1476589.1476686.
Retrieved from: http://doi.acm.org/10.1145/1476589.1476686

[22] Wolfram Research, Inc.: DAE (.dae). [online, 2019.04.19].
Retrieved from: https://reference.wolfram.com/language/ref/format/DAE.html

Appendix A

CD Structure

Structure of the files in the attached CD:

- **Animated Objects** — folder containing the source files of the application
- **Knight** — folder containing source files of the application that was part of the Excel@FIT conference
- **Script** — folder containing source files of the script used to convert the files into Xcode format
- **LaTeX** — folder containing the source of the bachelor thesis
- **Media** — folder containing a poster and a video
- **README** — file containing the CD structure